A TRAVELING SALESMAN ALGORITHM

William Willerson White

United States
Naval Postgraduate School

THESIS

A TRAVELING SALESMAN ALGORITHM

by

William Willerson White

October 1969

A Traveling Salesman Algorithm

by

William Willerson White
Captain, United States Army
B.A., The University of Texas, 1963

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
October 1969

TABLE OF CONTENTS

I. INTRODUCTION

A well-known programming problem is the traveling-salesman problem. As classically stated, a salesman must start from his hometown, travel to each of n-1 other cities, and return home. He is required to enter each of the n cities exactly once and leave each city exactly once, with the understanding that his entire trip is one continuous circuit. There is no requirement that the distances between cities be symmetric, that is, the distance from city i to city j need not be the same as that from j to i. The salesman's objective is to select a sequence of cities in such a way as to minimize the total distance traveled.

There are several algorithms which solve the traveling-salesman problem, but none offers the last word in terms of speed and ease of computation. The purpose of this investigation was to examine a newly proposed solution procedure developed by Professor Harold Greenberg.[1] His algorithm looked very promising in terms of hand computation on small problems, but this study focused on its desirability for handling larger problems, such as the forty-city case. The greatest effort went into programming the algorithm for the computer, then letting the computer serve as the primary tool of analysis.

---

[1]Greenberg, Harold, Professor, Department of Operations Analysis, Naval Postgraduate School, Monterey, California

A. DEFINITIONS

The distance (cost) between city i and city j is denoted c_{ij} for i = 1,...,n and j = 1,...n. In general, c_{ij} need not equal c_{ji}. The decision variable x_{ij} equals one if the salesman is to travel directly from city i to city j and equals zero, otherwise. Since traveling directly from city i to city i is not meaningful, x_{ii} must always equal zero for all i. Most solution procedures meet this requirement by assigning arbitrarily large values to c_{ii} for all i.

Associated with each city is the number of steps required by a solution to arrive at that city. For example, if the salesman is to go from city A to B to C and back to A, he arrives at B in one step, C in two steps, and back to A in three steps. Unless otherwise stated, the following discussion takes city one to be the city of origin and city n to be the last city visited prior to returning to city one.

A solution must provide a tour in order to be a traveling-salesman solution. A tour is defined as a sequence of transitions from city one through all other cities and back to city one, such that each city is entered exactly once and left exactly once, and it must take exactly n steps to return to city one. The latter clause guarantees there are no discontinuities in a tour. Deleting this clause yields the definition of a subtour. It is possible for a candidate solution to contain as many as n/2 subtours and still meet the requirement that each city is entered and left exactly once. To illustrate the concept of tours and subtours, consider these solutions to the four city problem: city 1 to 3 to 4 to 2 to 1 provides a tour; whereas city 1 to 2 to 1 and city 3 to 4 to 3 provides two subtours.

B. ALTERNATE FORMULATIONS

The traveling-salesman problem is essentially a combinatorial problem and there is no one way to solve it. The most direct approach is to select a starting point, evaluate the distance (cost) over each of the (n-1)! possible tours, and then choose the tour(s) with minimum total cost. A dynamic-programming formulation reduces the number of tours one must consider but the tables required can quickly exceed computer storage capacity. Branch-and-bound techniques can also be employed but suffer from similar limitations as dynamic programming. A fourth possibility might be to construct a solution procedure using an algorithm for finding minimal-cost flows in a network.

The traveling-salesman problem can be stated as an integer programming problem. Hadley [Ref. 6] and Dantzig [Ref. 1] present essentially the same formulation. Dantzig lets $x_{jkt} = 1$ or 0 according to whether or not the salesman travels from city i to city j on the t^{th} step, where $i,j,t = 1,\ldots,n$. Then, defining $x_{i,j,n+1} \equiv x_{ij1}$, he states the problems as follows:

minimize $$\sum_{i,j,t} c_{ij} x_{ijt} = z$$

subject to $$\sum_{i} x_{ijt} = \sum_{k} x_{j,k,t+1} \quad (j,t = 1,\ldots,n)$$

and $$\sum_{j,t} x_{i,j,t} = 1 \quad (i = 1,\ldots,n)$$

and $$x_{ijt} = 0 \text{ or } 1 \text{ for all } i,j,t.$$

The first set of constraints requires that if the salesman arrives at city j in t steps, he must leave that city on step t+1. The second sets requires that the salesman leave each city i once and only once. Solving this system of $n^3 + 1$ variables and $n^2 + n$ equations with an integer programming algorithm yields a minimum-distance tour.

The integer programming approach has several shortcomings. The third subscript t on x greatly increases the number of variables, and the number of constraints in the formulation is very large. Furthermore, integer programming algorithms normally require many more iterations than ordinary linear programming techniques.

There is a general class of indirect solution procedures that takes advantage of a special feature of the traveling-salesman problem. If one temporarily ignores the fact that a solution must provide a tour, the salesman's task can be formulated as a simple assignment problem, perhaps the most readily soluble of all linear programs.

The assignment formulation is achieved by letting $x_{ij} = 1$ or 0 according to whether or not the salesman is to travel from city i to j. Let c_{ii} be arbitarily large for all i. Then,

$$\text{minimize} \quad \sum_i \sum_j c_{ij} x_{ij}$$

$$\text{subject to} \quad \sum_i x_{ij} = 1 \quad (j + 1,\ldots,n)$$

$$\text{and} \quad \sum_j x_{ij} = 1 \quad (i = 1,\ldots,n)$$

Notice that there are only n^2 variables and 2n constraints. One constraint may be dropped because exactly 2n-1 are linearly independent. Furthermore, since the assignment problem is a form of the transportation problem, and since all restraining expressions equal one, any solution should yield 0 or 1 values for the x_{ij}'s /Ref. 2/. Thus, there is no need to add the integer constraint to the formulation above. Possible solution procedures include the simplex method, the

familiar transportation tableau, or an algorithm for finding minimal-cost flow in a network /_Ref. 4/.

An assignment outcome does not necessarily provide a traveling-salesman solution, but it does satisfy the requirements that each city is entered exactly once and departed exactly once and that $x_{ii} = 0$ for all i. Clearly, since the salesman's problem includes the added constraint that any set of values for the x_{ij}'s must provide a tour, his minimum-distance solution must be at least as great as the minimum assignment distance.

The so-called indirect procedures, alluded to above, typically use an assignment solution as a starting point. The optimal value of the assignment objective function provides a lower bound to any traveling-salesman solution. A non-optimal assignment solution may still provide a lower bound while being closer in magnitude to the salesman's minimum total distance. In the sense that it is closer, the latter lower bound is said to be "better" than the former. In practice, however, it is difficult to find a value larger than the optimal assignment solution and prove that it is, in fact, a lower bound to the traveling salesman problem.

Professor Greenberg's algorithm uses the optimal assignment solution as its point of departure because it provides a guaranteed lower bound and because special use is made of the coefficient vectors in the final simplex tableau. After the solution of the initial assignment problem, the algorithm simply manipulates the coefficient vectors until a minimum-cost tour is achieved. This procedure is explained in detail in the next chapter.

W. L. Eastman also begins with the optimal assignment solution and if it should provide a tour, then the problem is solved. If, in the more likely event, it does not, he selects the subtour with the least number of variables and sets each equal to zero, one at a time. For each case, he resolves the original assignment problem but with the added constraint. In practice this can be achieved by setting $c_{ij} = \infty$ corresponding to the x_{ij} required to be zero. If a tour is not obtained, he branches from that assignment problem with the smallest-valued objective function. Again the least-membered subtour is chosen to create new assignment problems, and so forth, until a tour is achieved. /Ref. 8/

J. D. C. Little and others have devised a branch-and-bound algorithm that starts with a lower bound, not to the traveling salesman problem, but to the assignment problem. Instead of obtaining the optimal solution to the assignment problem at each step, the smallest element is subtracted from each row of the cost matrix, and then the smallest element from each column of the result. The sum of the remaining elements is said to be a valid lower bound on the optimal assignment solution. At each branching step two new problems are created corresponding to $x_{ij} = 0$ and $x_{ij} = 1$. The x_{ij} chosen for each branching step is the one that yields as large a bound as possible when set equal to zero. Branching proceeds until a tour is obtained. Little's method normally requires many more branches than does Eastman's, but there is less computation at each step. Some performance results are available on Little's algorithm. The mean execution time on an IBM 7090 computer for 100 thirty-city problems was 58.5 seconds and the average for five forty-city problems was 8.37 minutes. /Ref. 9/

C. PLAN OF THE REPORT

Chapter II presents a detailed description of the solution procedure suggested by Professor Greenberg. His proposed algorithm is complete in the sense that the criteria for proceeding from one step to the next are completely specified and guarantee an optimal solution. In some instances, however, it is not readily apparent how one can most efficiently meet these criteria. Chapter III deals with these questions of technique as it describes how the algorithm was programmed for the computer. When there were alternate computational approaches possible, an attempt was made to find a good one. There is no claim that the particular techniques devised to meet the algorithm criteria are the ones Professor Greenberg, himself, would have chosen. Chapter IV outlines the salient results of computational experience with the computer program. The algorithm and the program are evaluated in the light of this experience.

II. THE ALGORITHM

An algorithm is defined by Webster as "a rule or procedure for solving a mathematical problem that frequently involves repetition of an operation." This chapter seeks to present Professor Greenberg's algorithm for solving the traveling-salesman problem in the logical sequence of its development, concluding with a concise statement of the algorithm, itself.

A. THE TRAVELING-SALESMAN PROBLEM

A traveling salesman is faced with the problem of finding the shortest route (or the least-cost route) through n cities. Furthermore, he must meet the following conditions: 1) he must enter each city once and only once, 2) he must depart from each city once and only once, and 3) his route must be a tour, that is, there must be no discontinuities in his route. There are (n-1)! possible solutions to this problem, since the salesman can go to any of n-1 cities on his first step, then to any of the remaining n-2 cities on his second step, and so forth, until he returns to his starting point on the n^{th} step. Optimal solutions, of course, are found within this set of (n-1)! tours.

Clearly, the above is merely an allegorical statement of a problem that has relevance to a variety of situations. For example, suppose fixed electronic components must be connected in a continuous circuit such that the least amount of wire is required. Disregard of conditions (1) and (2) might leave some components out of the circuit entirely or, on the other hand, cause short circuits. Violation of

condition (3) would result in two or more subcircuits, with current flowing in only one.

B. THE ASSIGNMENT FORMULATION

Modifying the traveling-salesman problem by disregarding the third requirement above, yields a problem which is readily adaptable to the assignment formulation, perhaps the most easily soluble of all programming problems. The assignment problem is usually stated in terms of assigning n men to n jobs in such a way that overall productivity is maximized, assuming that a productivity index can be associated with each man-job combination. There are n! possible solutions to this problem, since the first man can be assigned to any of the n jobs, the second man to any of the remaining n-1 jobs, and so forth.

Dropping the requirement, then, that a solution must provide a tour, the modified salesman problem is to assign n cities to n cities (the same n cities) such that the sum of city-to-city distances is minimized. The possibility of assigning city i to itself is precluded by giving each c_{ii} an arbitrarily large value. The possibility of other types of subtours appearing in a solution, however, cannot be restricted without destroying the assignment formulation.

Since no city is ever assigned to itself in the modified salesman problem, the size of the solution set is smaller than n!, the usual size for assignment problems. The table below illustrates solution-set sizes for several selected problems:

Number of cities (n)	Number of Assignment Solutions (n!)	Number of Modified Traveling-Salesman Solutions $n! \geq (?) \geq (n-1)!$	Number of Solutions that are Tours (n-1)!
2	2	1	1
3	6	2	2
4	24	9	6
5	120	44	24

A generalized expression for the third column above was not derived, but it is clear that the fraction of the time a modified salesman solution is also a tour is greater than $(n-1)!/n! = 1/n$.

An optimal solution to the modified salesman problem provides a lower bound on the minimum-distance solution to the original problem. This is true, because the modified problem differs from the original only in that it has fewer constraints. If the optimal assignment solution to the modified problem provides a tour, then the original problem is solved; if not, further steps must be taken.

To express the assignment formulation in the notation of Chapter I, let $x_{ij} = 1$ or 0 according to whether or not the salesman is to travel from city i to city j; $i,j = 1,\ldots,n$. Let c_{ij} be the distance between city i and j for all i and j. Set c_{ii} arbitrarily large for all i. Then,

minimize $$\sum_i \sum_j c_{ij} x_{ij}$$

subject to $$\sum_i x_{ij} = 1 \quad (j=1,\ldots,n)$$

and $$\sum_i x_{ij} = 1 \quad (i=1,\ldots,n)$$

and $$x_{ij} \geq 0 \text{ for all } i,j$$

Alternately, this formulation can be expressed in matrix notation as follows:

minimize $\quad c^T x = z$

subject to $\quad Ax = b$

and $\quad x \geq 0$

where the definitions of c^T, x, b, and A can be indicated most conveniently by example. Consider the three-city problem:

$$c^T = (c_{11}, c_{12}, c_{13}, c_{21}, c_{22}, c_{23}, c_{31}, c_{32}, c_{33}),$$

$$x = \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{31} \\ x_{32} \\ x_{33} \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

$$\text{and } A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

In general, c^T is $1 \times n^2$, x is $n^2 \times 1$, b is $n^2 \times 1$, and A is $2n \times n^2$. Notice that since the k^{th} column of A corresponds to the k^{th} x_{ij} of x.

the i^{th} element and the $(n+j)^{th}$ element of the k^{th} column of A equal one and all other elements in that column equal zero.

The assignment problem has several important features that are listed below without proof. (see references 3 and 5 for proofs.)

1. The assignment problem has a feasible solution.

2. The rank of the matrix A is 2n-1, thus any row may be deleted as redundant.

3. Any set of 2n-1 linearly independent column vectors from A forms a basis B.

4. The inverse of any basis B is composed entirely of zero, one and minus-one elements.

5. Any basic feasible solution to Ax = b is degenerate. In fact, there are exactly n basic variables equal to one and exactly n-1 equal to zero, assuming all nonbasic variables are arbitrarily set equal to zero.

C. THE SIMPLEX SOLUTION OF THE ASSIGNMENT FORMULATION

There are alternate methods for solving the assignment problem but the following discussion assumes some variant of the simplex method is chosen. The final simplex tableau contains not only the optimal solution to the assignment problem, but other interesting information, as well. Before proceeding, a few more notational conventions and definitions are stated below:

1. Redefine the coefficient matrix A such that one row (any row) is deleted.

2. Temporarily redefine the vector x such that the subscripts on its elements run from 1 to n^2 rather than 1,1 to n,n.

3. Make the same change in subscript notation for the elements of c^T.

4. Denote the j^{th} column of A as a_j for all $j = 1,\ldots, n^2$.

5. Let B denote a basis in A, that is, any set of 2n-1 column vectors in A.

6. Let x_B denote the vector of x_j's corresponding to the a_j's in B. Call the elements of x_B the values of the basic variables. Note, since $x_B = B^{-1}b$, x_B is simply the vector obtained by summing each row of B. Requiring $x_B \geq 0$, then, insures its elements are ones and zeros, exclusively.

7. Let c_B^T denote the vector of c_j's corresponding to the elements in x_B.

8. Define $y_j \equiv B^{-1}a_j$. Note that the y_j's are composed entirely of zero, one, and minus-one elements.

9. Define the relative cost for x_j as $\bar{c}_j \equiv c_j - c_B^T y_j$.

The simplex procedure involves finding a basis B such that $x_B = B^{-1}b$ minimizes z. A feasible solution is recognized when the elements of x_B are zeros and ones, exclusively, and an optimal solution, when $\bar{c}_j \geq 0$ for all $j = 1,\ldots,n^2$.

Notice that $\bar{c}_j = 0$ always for all x_j's in x_B because for such x_j's, $c_j = c_B^T y_j$. If $\bar{c}_j = 0$ at optimality for a nonbasic x_j, then an alternate optimum can be obtained by bringing that x_j into the basis. In this case, the simplex methodology specifies which basic variable is to be replaced in order to maintain feasibility.

To appreciate the impact of the relative costs on the value of z, consider the following relationship:

$$\hat{z} = z + \sum_{j \in R} \bar{c}_j x_j$$

where $\hat{z}$ is the new value of z and R is the index set of all nonbasic variables. This equation shows that if nonbasic variables are set equal to one, z is increased by the sum of their relative costs.

Of course, if one or more nonbasic variables are set equal to one, some change must take place in the x_B vector. This is true since an assignment solution requires exactly n variables equal to one, including the nonbasic variables. In order to measure such effects on x_B, consider the following equality:

$$\hat{x}_B = x_B - \sum_{j \in R} y_j x_j$$

where $\hat{x}_B$ is the new set of values for x_B. Normally $x_j = 0$ for all j in R, but if one or more such nonbasic variables are set equal to one, the relation above changes the values of the basic variables. Setting one or more x_j's not in x_B equal to one is said to be constraint feasible as long as the elements of $\hat{x}_B$ are zeros and ones, exclusively.

D. STATEMENT OF THE ALGORITHM

The algorithm under study begins with the optimal assignment solution to the modified traveling-salesman problem. If the optimal assignment solution provides a tour, then the original problem is solved. Otherwise, the algorithm begins to introduce combinations of nonbasic variables at the one level until a tour is achieved. Complexities arise in the measures taken to assure that combinations are considered in ascending order of their relative costs. If these steps are not taken, there is a possibility that nonbasic variables not yet tried yield lower-cost tours.

In the light of the background developed in the previous sections, the algorithm is as follows:

1. Drop the constraint that a traveling-salesman solution must provide a tour.

2. Solve this modified problem as an assignment problem using a variant of the simplex method.

3. Check the optimal assignment solution for a tour. If a tour is provided, stop. Otherwise, continue.

4. Drop simplex tableau vectors corresponding to the original x_{ii} variables. Drop tableau vectors corresponding to basic variables. There are n^2-3n+1 vectors remaining. Let $m = n^2-3n+1$.

5. Rank tableau vectors in ascending order of their respective relative costs.

6. Let $x_j = x_i, y_j = y_i$, and $\bar{c}_j = \bar{c}_i$ where $i = 1$ if $\bar{c}_j$ is first in the ranking, $i = 2$ if $\bar{c}_j$ is second, and so forth, through $i = m$ if $\bar{c}_j$ is last in the ranking. (Retain information required to reconvert subscripts as needed when checking for tours.)

7. Let $i = 0$.

8. Increment i by one.

9. If $x_i = 1$ is not constraint feasible and $i = 1$, return to step 8. If $x_i = 1$ is not constraint feasible and $i > 1$, go to step 11. If $x_i = 1$ is constraint feasible, continue.

10. If $x_i = 1$ and $\hat{x}_B = x_B - y_i$ provide a tour, go to step 21. Otherwise, return to step 8 if $i = 1$; continue if $i > 1$.

11. Generate all combinations of $x_i = 1$ with $x_1 = 1$, $x_2 = 1, \ldots,$ $x_{i-1} = 1$. (Note this is construed to mean $x_i = 1$ appears in each combination along with one or more of the other variables indicated set equal to one. There are $2^{i-1} - 1$ such combinations.)

12. Disregard all combinations with relative costs greater than $\bar{c}_{i+1}$. (Define $c_{m+1} = \infty$.)

13. Disregard all remaining combinations that are not constraint feasible.

14. Check remaining combinations, in ascending order of their relative costs, for tours. As soon as a combination is found to provide a tour, go to step 21. If no combination yields a tour, continue.

15. If i = 2 and/or $\bar{c}_{i+1} = \bar{c}_i$, return to step 8. Otherwise, continue.

16. Let k = i.

17. Decrement k by one.

18. Generate all combinations of $x_k = 1$ with $x_1 = 1$, $x_2 = 1,\ldots,$ $x_{k-1} = 1$.

19. Same as steps 13, 14, and 15.

20. If $k > 2$, return to step 17. If k = 2, return to step 8.

21. If coming from step 19, let i = k. Otherwise, do not change i.

22. Compute total distance for the tour found. Letting K be the index set of nonbasic variables included in the solution, the total distance for this tour is $\hat{z} = z + \sum_{k \in K} \bar{c}_k$.

23. If $i \leq 2$, go to step 32. Otherwise, continue.

24. If all combinations of nonbasic variables preceding x_i have already been checked against the current value of $\hat{z}$-z as an upper bound, go to step 32. Otherwise, continue.

25. Decrement i by one.

26. Generate all combinations of $x_i = 1$ with $x_1 = 1$, $x_2 = 1,\ldots,$ $x_{i-1} = 1$.

27. Disregard all combinations with relative costs greater than $\hat{z}$-z.

28. Disregard all remaining combinations that are not constraint feasible.

29. Check remaining combinations, in ascending order of their relative costs, for tours. As soon as a combination is found to provide a tour, go to step 30. If no combination yields a tour, go to step 31.

30. Update the value of $\hat{z}$ for the latest tour found and continue.

31. If $i > 2$, return to step 25. Otherwise, continue.

32. The latest tour found and its associated $\hat{z}$ is an optimal traveling-salesman solution. Stop.

Since eventually all combinations of nonbasic variables are considered, the algorithm must eventually find a tour. Suppose a tour is first found using $x_i = 1$ in some combination with the preceding nonbasic variables. This tour's relative cost is less than or equal to $\bar{c}_{i+1}$ because only combinations meeting this criterion were considered. Thus, there is no need to try combinations beyond x_i. There is a need, however, to reconsider combinations prior to x_i. For example, $x_4 = 1$ in combination with $x_1 = 1$ may have been rejected earlier because its relative cost is greater than $\bar{c}_5$, but it may be less than $\hat{z}-z$. Thus, proceeding backward through the nonbasic variables using the current value of $\hat{z}-z$ as an upper bound guarantees the minimum $\hat{z}$ will be found.

III. THE COMPUTER PROGRAM

The algorithm was programmed for the IBM System/360 Model 67 Operation System using FORTRAN IV (H level) programming language /Ref. 7/. A listing of the complete program is provided in Appendix A.

Several routines used in the program were not developed as part of this study but were modified for use here as needed. Subroutines ASSIGN, LIP, ELEM, and PIVOT were supplied by Professor Greenberg and are used to solve the initial assignment problem. Subroutine RAND is a simple routine for generating city-to-city distances, uniformly distributed from 0 to 100. The distance from each city to itself is arbitrarily set equal to 999. Subroutine TIMEIT is used to measure algorithm execution times in seconds. Part of TIMEIT appears in assembly language.

The following list gives a brief description of the various routines developed especially for this study.

1. The Main Program follows the sequence of algorithm steps as stated in Chapter II. Provision is made to either read in a cost matrix or to generate one randomly. Other provisions are made to abort if something unexpected goes wrong and to begin execution on the next problem.

2. Subroutine PRINTC writes out the cost matrix. Also, alternate storage space is provided for the costs because they get overlayed in the assignment routines by the relative costs.

3. Subroutine PRINTA writes the optimal assignment solution in tableau form. If the problem is larger than seven cities they y_j

column vectors are not printed because there are too many to fit on a page. Subroutine PRINTA is also used again to print the optimal assignment solution after artificial variables remaining in the basis have been replaced.

4. Subroutine SOLCHK checks constraint feasible solutions for tours. If a tour is found, it is printed.

5. Subroutine RANKC ranks the relative costs in ascending order of magnitude. Actually, the costs are not moved around in storage, but second-order subscripts are assigned to indicate the desired ranking.

6. Subroutine REPART was devised to replace any artificial variables left in the assignment solution and to select a redundant row for deletion from the final tableau.

7. Subroutine LINCOM is the most important subroutine in terms of the algorithm at hand. This subroutine generates the combinations of the nonbasic variables, compares the relative cost of each combination with an upper bound, and checks for constraint feasibility.

A. DEFINITIONS

Three common blocks (storage areas) are used in the program. Common Block ASGBLK is used exclusively with the four assignment routines. Common Block SHARE is used to pass information from the assignment routines and from Subroutine RAND to the traveling-salesman algorithm routines. Common Block TSBLK is used to pass values among the algorithm routines. For easy reference, this section defines the variables in common in the order of their appearance. Variables in ASGBLK are not listed because they are only needed within the four assignment routines which are not part of this study.

1. Common Block TSBLK

a. COST is the sum of the relative costs included in a particular combination of nonbasic variables, each taken at the one level.

b. CSTLC(1600) is initially used as alternate storage for the city-to-city costs. After artificial variables have been replaced, it is used to store relative costs of combinations in ascending order of magnitude.

c. NLC(1000) is used to store a combination identifying number corresponding to each value in CSTLC(1600). With these numbers, combinations can be regenerated as needed.

d. IC(1600) is used to store the subscripts of the variables associated with the ranked, relative costs. For example, the first value in IC(1600) is the subscript of the nonbasic variable with the smallest relative cost.

e. KC(40) is used in Subroutine PRINTC to write out the city-to-city costs as integers.

f. KB(120) is used to store the new values of the basic variables when one or more nonbasic variables are set equal to one. The one-values for the nonbasic variables are augmented to this vector starting after the last basic-variable value. (Also, see description of IB(120) below.)

g. IAX(40) is used in Subroutine PRINTA to write out a row of the final assignment tableau.

h. IROW(40) and ICOL(40) are used in Subroutine SOLCHK to store the first and second subscripts, respectively, of variables after these have been reconverted from single-subscript form.

i. LC(40) is used to store a binary number up to 40 digits in length. If the k^{th} digit is a one, then the k^{th} ranked, nonbasic variable is to be included in the particular combination at the one level. If the k^{th} digit is zero, then the k^{th} ranked, nonbasic variable is not to be included. LC(40) can be used to represent up to $2^{40} - 1$ different combinations, one at a time.

j. KZ is the truncated value of Z in Subroutine PRINTA. In Subroutine SOLCHK, KZ is the truncated value of XZ = Z + COST. If the city-to-city costs are integers, KZ = XZ always.

k. NC = N-3MM+1 is the number of nonbasic variables, excluding x_{ii} variables and artificials.

l. J is used in the Main Program to index the progression through the ranked, nonbasic variables.

m. ICJ = IC(J).

n. L1 = LEG(1,ICJ) and L2 = LEG(2,ICJ).

o. J1 = J-1.

p. CST is the current upper bound against which combinations are compared. Any combination of nonbasic variables at the one level with a relative cost greater than CST is disregarded.

q. XZ = Z+COST. Subroutine SOLCHK truncates this value when writing it out. (See definition (j) above.)

2. Common Block SHARE

a. C(1600) is used initially to store the city-to-city costs. After the assignment solution, the relative costs are stored here. (Also, see description of CSTLC (1600) above.)

b. B(80,80) is the inverse of the optimal assignment basis.

c. H(80) is used to store the values of the basic variables in the optimal assignment solution. (Also, see description of KB(120) above.)

d. Z is the optimal assignment total cost.

e. LEG(2,1600) is a matrix of values used to convert single-subscripted variables back to their original double-subscripted form. For example, $x_k = x_{ij}$ if i = LEG(1,k) and j = LEG(2,k) - MM where MM is the number of cities. Furthermore, LEG provides a convenient device for obtaining the product $B^{-1} a_k = y_k$. It is known that the i^{th} and $(j+MM)^{th}$ elements of a_k are ones, if i and j are defined as above, and all other elements of a_k are zeros. Thus, rather than multiplying B^{-1} times a_k, simply add the i^{th} and $(j+MM)^{th}$ elements of each row of B^{-1} to produce the vector y_k.

f. IB(120) is used to store the subscripts of the optimal assignment basic variables. Negative subscripts indicate artificial variables. Starting just after the last basic-variable subscript, nonbasic-variable subscripts are appended to this vector for each nonbasic variable set equal to one in a particular combination. (Also, see description of KB(120) above.)

g. MM is the size of the problem, that is, the number of cities.

h. M is the number of constraining equations in the assignment formulation. Initially, M = 2(MM), but in Subroutine REPART it is reduced by one.

i. $N = (MM)^2$ is the total number of variables, excluding artificials.

B. DESCRIPTION OF SELECTED SUBROUTINES

The Main Program was written to conform closely to the algorithm as presented in Chapter II and can be followed readily from the listing in Appendix A. Further amplification of Subroutines LINCOM, SOLCHK, and REPART is considered necessary, however, in order to clarify the workings of the overall program. Subroutine REPART was devised only because the assignment routines used leave artificials in the basis and do not drop a redundant equation. Subroutines SOLCHK and LINCOM represent techniques created to fulfill algorithm requirements and, as such, are the most important products of this study.

1. Subroutine REPART

Subroutine REPART scans the subscripts of the basic variables returned from the assignment routines. As soon as a negative subscript is located, its corresponding artificial variable is replaced in the basis by the first, ranked, nonbasic variable that has a nonzero element in the corresponding row of its y_j vector. Using this nonzero element as a pivot element, standard row transformations are performed. The result is that the nonbasic variable is brought into the basis and the artificial is dropped. The optimal value of the objective function does not change because the artificial variable was necessarily at the zero level implying its replacement must also be at the zero level. After replacing an artificial, relative costs for the remaining nonbasic variables are recomputed and these variables are then ranked in ascending order of their new relative costs. The above process is repeated until all artificials have been removed from the basis.

It is known that one of the assignment-tableau rows is redundant. If, while trying to replace an artificial, a nonzero pivot element

cannot be found, that row is earmarked for deletion. Otherwise, after all artificials have been replaced, each row is scanned for all zero elements. If such a row is found, it is dropped. Otherwise, the last tableau row is arbitrarily selected for deletion. If other than the last row is dropped, remaining rows are shifted upward in the tableau to fill the gap.

Normally, control is returned to the statement following the calling statement in the Main Program. If for any reason, however, more than one row has gone to all zeros, control is returned to statement 10 in Main and execution on the next problem is started.

2. Subroutine SOLCHK

Subroutine SOLCHK examines an assignment solution to see if it provides a tour. Its approach is to count the number of steps required to return to the city of origin. If this number is less than the total number of cities MM, then subtours exists and control is returned to the statement following the calling statement in Main. If the number of steps required is exactly equal to MM, the solution is a tour. In the latter event, the solution is printed with its associated total distance and control is returned to the statement in Main indicated in the parenthesis of the calling statement.

To illustrate the bookkeeping involved, consider the two following four-city assignment solutions:

	Steps
$x_8 = x_{2,4} = 1$	1
$x_3 = x_{1,3} = 1$	
$x_{14} = x_{4,2} = 1$	2
$x_9 = x_{3,1} = 1$	

	Steps
$x_{10} = x_{3,2} = 1$	1
$x_{13} = x_{4,1} = 1$	3
$x_{8} = x_{2,4} = 1$	2
$x_{2} = x_{1,2} = 1$	4

Both are assignment solutions, but only the second provides a tour.

Notice that the routine will keep scanning the KB vector until the $(MM)^{th}$ one-value is found and then scanning is terminated. This approach is justified because only constraint feasible solutions are sent to SOLCHK and these solutions necessarily have exactly MM variables equal to one, including any nonbasic variables set equal to one.

3. Subroutine LINCOM

The essence of the algorithm lies in altering the optimal assignment solution by setting selected nonbasic variables equal to one until a tour is achieved. Subroutine LINCOM was devised to generate all the required combinations of nonbasic variables. An attempt was made to program LINCOM so that the number of combinations necessary to consider be held to a minimum. The objective was to generate them in ascending order of total relative costs so that as soon as a combination's cost exceeded the current upper bound, no further combinations need be considered. This objective was only partially achieved.

The most difficult problem lay in the fact that there is no way to completely predetermine the ranking of all possible combinations. Consider the case where $\bar{c}_1 \leq \bar{c}_2 \leq \ldots \bar{c}_5 \leq \bar{c}_6$ and $x_5 = 1$ is found not constraint feasible. Before testing $x_6 = 1$, all combinations of $x_5 = 1$ with the preceding nonbasic variables must be tried because some may have total relative costs less than or equal to $\bar{c}_6$.

One way to test the combinations would be to try $x_1 = 1$, $x_2 = 1$, and $x_4 = 1$ singly with $x_5 = 1$, then in pairs, then triplets, and finally all four. A better approach, however, is to set up the four-digit binary number 0000 and then simply increment it by one, fifteen times. Number the digit positions from one through four, from right to left. Then, if a one appears in the j^{th} position where $j = 1,\ldots,4$, let $x_j = 1$ appear in the combination. The possible combinations can now be represented as shown below:

Identifying Number	Combination
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Combination number 9, for example, represents $x_5 = 1$ with $x_4 = 1$ and $x_1 = 1$ for a total relative cost of $\bar{c}_5 + \bar{c}_4 + \bar{c}_1$.

Notice that the following generalizations can be made about the respective total relative costs of the combinations above:

$$\#1 \leq \#2 \leq \#3,$$

$$\#4 \leq \#5 \leq \#6 \leq \#7,$$

$$\#8 \leq \#9 \leq \#10 \leq \#11,$$

and $$\#12 \leq \#13 \leq \#14 \leq \#15.$$

Furthermore, #3 $\leq$ #5 $\leq$ #9, but #3's relationships with #4 and #8 cannot be stated, in general. Also, #7 $\leq$ #11 $\leq$ #13, but #7's relationships with #8, #9, #10, and #12 are unknown. Finally, #11 $\leq$ #13, but there is no way to tell in advance which is smaller, #11 or #12.

While there are always many organizational difficulties associated with writing and debugging a large program, the situation discussed above presented one of the most challenging theoretical problems to resolve. The scheme finally chosen is stated below:

If a combination's relative cost is greater than the current upper bound and the combination's identifying number is equal to

1) 2^i, where i is any nonnegative integer, terminate generation of further combinations (because all their relative costs will exceed the current upper bound), or

2) $2^i + 2^j$, where i and j are any nonnegative integers such that $j < i$, skip to combination number 2^{i+1} and continue generating, or

3) otherwise, continue generating.

In most cases, the above procedure greatly reduces the number of combinations necessary to generate. It could be expanded by saying that if the combination number is equal to $2^i+2^j+2^k$, where i, j, and k are nonnegative integers such that $k < j < i$, skip to number 2^i+2^{j+1}, and so forth. Except for very large problems, however, such added checks would probably consume more computer time than they would save.

Another facet of the problem of trying to generalize the relationships among relative costs is presented when a combination is found constraint feasible. To illustrate the situation, suppose combination number 11 is found to provide a relative cost less than the current upper bound and is also found constraint feasible. If #11 is sent to SOLCHK at this point and found to provide a tour, there can be

no certainty that it is a least-cost tour among the current set of all possible combinations. This is because #12, as yet unchecked, may provide an even smaller relative cost, may be constraint feasible, and may also provide a tour.

The technique devised to resolve the above difficulty was to store the identifying number and relative cost of each combination found constraint feasible. With each addition to storage, the numbers and relative costs are reranked in ascending order of relative costs. Finally, when all necessary combinations have been generated and checked, the constraint feasible solutions are sent to SOLCHK in ascending order of relative costs, thereby insuring the first tour found has a cost less than or equal to any tour the remaining solutions in storage might provide. As soon as a tour is found, control is returned to the first address indicated in the parenthesis of the calling statement in Main. If none of the solutions in storage provides a tour, control is returned to the statement in Main following the calling statement.

Finally, a word about the regeneration of combinations required by the above procedure. The decimal identifying number in storage is converted to binary through the standard technique of successive divisions by 2. Remainders become the binary digits from right to left. The binary number, then, is the original representation of the combination represented temporarily in storage by the decimal identifying number. Using the binary number, the combination of interest is reconstructed and sent to SOLCHK along with its associated relative cost.

C. LIMITATIONS OF THE PROGRAM

The storage areas of the program are set up to deal with problems smaller than 41 cities, but these areas can be expanded easily if necessary. The four subroutines used to provide an optimal assignment solution are programmed to terminate execution in the unlikely event that more than 299 iterations are required. The only other significant limitations of the program are found in Subroutine LINCOM, but one of these presents a serious restriction.

In Subroutine LINCOM, up to 1000 constraint feasible solutions can be stored at one time, but this does not appear to present an important limitation. In any case, storage space for CSTLC and NLC in Common Block TSBLK can be increased considerably if necessary. The most serious restriction and the one hardest to correct is the fact that Subroutine LINCOM can generate all the possible combinations for no more than 32 nonbasic variables. This means that if a tour is not found upon setting the 33^{rd} ranked, nonbasic variable equal to one, then execution on this problem must terminate. The reason for this situation is that each combination must have a unique identifying number associated with it in LINCOM and the largest integer number that can be expressed on the IBM 360 is $2^{31}-1$. There are $2^{32}-1$ possible combinations of the 33^{rd} variable with the 32 preceding it, hence, the problem. As an example of the gravity of the restriction, note that a forty-city problem presents 1481 nonbasic variables for consideration.

The perplexity above is not insurmountable, but it was not resolved in this study. The possibility of representing the identifying number in floating-point mode is precluded because a form of the

number is used as a DO Loop parameter. (See the variable KT in LINCOM) It might be possible to represent the number as an element of a vector. For example, each of the first 2^{31}-1 numbers might be prefixed with a 1, the second set of 2^{31}-1 numbers with a 2, and so forth. The DO Loop using the number as a parameter could, in like manner, be expanded into a double DO Loop.

IV. COMPUTATIONAL EXPERIENCE

Unfortunately, time constraints on this study did not permit extensive testing of the computer program in its final form. Enough was accomplished, however, to indicate several important conclusions.

Appendix B presents a partial facsimile of the program output for the same ten-city problem that is discussed in References 8 and 9. Eastman's algorithm required the solution of eleven assignment problems before the optimal tour was obtained. Little's method required 31 branches to arrive at the desired solution. Appendix B indicates the programmed algorithm under study took only 1.50 seconds after the initial assignment solution to obtain the optimal tour. The first tour was found after trying combinations of the first 23 ranked, nonbasic variables and this tour turned out to be the optimal one.

City-to-city costs were generated by Subroutine RAND for 17 twenty-city problems. Fourteen of these problems were solved by the program in a mean time of 25 seconds and after trying an average of 15 nonbasic variables for each problem. Execution was terminated on the other three problems because a tour had not been found within the first 33 nonbasic variables.

Ten randomly generated thirty-city problems were attempted and nine were solved in an average time of 56 seconds. The mean number of nonbasic variables required for testing in these nine problems was 16.

Finally, Subroutine RAND was used to create three forty-city problems. The first ran .07 seconds after the assignment solution to discover this was also an optimal tour. The second problem ran 549

seconds before terminating after trial of the 33^{rd} nonbasic variable failed to provide a tour. The third problem was solved in 187 seconds after trying only four nonbasic variables.

In view of the computational experience cited above, one can conclude that the algorithm works and that the program works with one exception. Of the 31 problems tried, five could not be solved because tours had not been found upon setting the 33^{rd} nonbasic variables equal to one. If the program is to be of general use, this somewhat arbitrary restriction in Subroutine LINCOM must be removed. The limitation could be withdrawn if a way is found to obviate the identifying numbers presently associated with each possible combination. Since $2^{31}-1$ is the largest integer that can be represented on the IBM System/360, none beyond the 32^{nd} ranked, nonbasic variable can be tried in combination with its preceding variables.

Several steps can be taken to increase the execution speed of the program. All write statements, except those in Subroutine SOLCHK and the various termination messages throughout the program, can be replaced with CONTINUE statements. In subroutine REPART there is no particular reason why relative costs are recomputed and reranked after each artificial replacement. These two chores could be postponed until all artificials have been replaced. Since a replacement variable enters the basis at the zero level, it is not necessary that its relative cost be the least possible.

In conclusion, a possible improvement to the algorithm, itself, is suggested. The algorithm is very efficient if a tour is found within the first few nonbasic variables tried, otherwise much time is consumed rechecking combinations already considered against a new upper bound,

and perhaps again against an even newer upper bound, and so forth. This phenomenon can be observed in the example output displayed in Appendix B. The algorithm might be improved as follows: 1) obtain a relatively low-cost tour by linking together, on some reasonable fashion, the subtours appearing in the optimal assignment solution, 2) compute $\hat{z}$ for this tour, 3) using $\hat{z}$-z as an upper bound, begin with the last, ranked, nonbasic variable whose relative cost is less than $\hat{z}$-z and proceed backward just as in the original algorithm after the first tour is found. In this manner, it is not necessary to proceed forward through the ranked, nonbasic variables looking for that first tour, and it is necessary to work backwards through the variables only once.

```
C  PLACE GREEN JOB CARD HERE.
//  EXEC  FORTCALG,REGION.GO=160K,TIME.GO=10
//FORT.SYSPRINT DD SPACE=(CYL,(4,1))
//FORT.SYSIN DD *
C  A TRAVELING-SALESMAN ALGORITHM
      COMMON /TSBLK/ COST,CSTLC(1600),NLC(1000),
     1               IC(1600),KC(40),KB(120),IAX(40),
     2               IROW(40),ICOL(40),LC(40),
     3               KZ,NC,J,ICJ,L1,L2,J1,CST,XZ
      COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
     1               LEG(2,1600),IB(120),MM,M,N
C  MM IS THE NUMBER OF CITIES.
C  NRAND=1 IF COSTS ARE TO BE RANDOMLY GENERATED.
   10 READ(5,500,END=90) MM,NRAND
  500 FORMAT(2I2)
      M=2*MM
      N=MM*MM
      IF(NRAND.EQ.1) CALL RAND(&20)
      READ(5,510) (C(J),J=1,N)
  510 FORMAT(16F5.0)
   20 CALL PRINTC
C  OBTAIN OPTIMAL ASSIGNMENT SOLUTION.
      CALL ASSIGN(&10)
      CALL PRINTA
C  START CLOCK.
      CALL TIMEIT(0,TIME)
C  SEE IF OPTIMAL ASSIGNMENT SOLUTION IS A TRAVELING-
C  SALESMAN SOLUTION.  IF YES GO TO 80, IF NO CONTINUE.
      COST=C.
      CALL SOLCHK(&80)
C  RANK NONBASIC VECTORS IN FINAL TABLEAU IN ASCENDING
C  ORDER OF THEIR ASSOCIATED RELATIVE COSTS.
      CALL RANKC
C  REPLACE ARTIFICIALS REMAINING IN OPTIMAL ASSIGNMENT
C  SOLUTION AND DROP A REDUNDANT EQUATION FROM THE TABLEAU.
C  NOTE THAT M IS DECREMENTED BY 1 IN SUBROUTINE REPART.
      CALL REPART(&10)
      CALL PRINTA
      C(IC(NC+1))=1000000.
      CSTHLD=C.
C  NC=MM*MM-3*MM+1 IS THE TOTAL NUMBER OF NONBASIC
C  VARIABLES OF INTEREST.
C  PROCEED FORWARD THROUGH THE RANKED, NONBASIC VARIABLES.
      DO 50 L=1,NC
      J=L
      ICJ=IC(J)
      KOST=C(ICJ)
      WRITE(6,610) ICJ,KOST
  610 FORMAT(//10X,'TRY X(',I2,') = 1   REL COST = ',I3)
      L1=LEG(1,ICJ)
      L2=LEG(2,ICJ)
      IB(M+1)=ICJ
      KB(M+1)=1
      COST=C(ICJ)
      DO 30 I=1,M
      XHOLD=H(I)-B(I,L1)-B(I,L2)
      KB(I)=XHOLD+SIGN(.5,XHOLD)
C  IF X(ICJ)=1 IS NOT CONSTRAINT FEASIBLE, GO TO 40.
      IF(KB(I).NE.0.AND.KB(I).NE.1) GO TO 40
   30 CONTINUE
      CALL SOLCHK(&58)
   40 IF(J.EQ.1) GO TO 50
C  TRY ALL POSSIBLE COMBINATIONS OF X(ICJ)=1 WITH THE
C  PRECEDING NONBASIC VARIABLES.  DISREGARD ANY COMBINATION
C  WITH RELATIVE COST GREATER THAN CST=C(IC(J+1)).
      J1=J-1
      WRITE(6,620) J1
  620 FORMAT(10X,'TRY THE (2**',I2,') - 1 LN COM')
```

```
      CST=C(IC(J+1))
C  IF A COMBINATION PROVIDES A TOUR GO TO 58,
C  OTHERWISE CONTINUE.
      CALL LINCOM(&58,&10)
C  IF CST IS EQUAL TO THE PREVIOUS CST GO TO 50.
      IF(CST.EQ.C(ICJ)) GO TO 50
      IF(J.EQ.2) GO TO 50
C  PROCEEDING BACKWARD THROUGH THE RANKED, NONBASIC
C  VARIABLES, RECHECK OLD COMBINATIONS AGAINST THE NEW CST.
      CSTHLD=CST
      KJ1=J-1
      DO 45 K=2,KJ1
      J=KJ1-K+2
      J1=J-1
      ICJ=IC(J)
      L1=LEG(1,ICJ)
      L2=LEG(2,ICJ)
      IB(M+1)=ICJ
      KB(M+1)=1
      WRITE(6,630) ICJ
  630 FORMAT(//10X,'RETURN TO X(',I2,') AND')
      WRITE(6,620) J1
C  IF A COMBINATION PROVIDES A TOUR GO TO 60,
C  OTHERWISE CONTINUE.
      CALL LINCOM(&60,&10)
   45 CONTINUE
   50 CONTINUE
C  THE PROGRAM SHOULD NEVER REACH THIS POINT BECAUSE
C  THERE ARE (MM-1)! POSSIBLE TOURS.
      CALL TIMEIT(-1,TIME)
      WRITE(6,640)
  640 FORMAT(//10X,'A TRAVELING SALESMAN SOLUTION',
     1           ' DOES NOT EXIST')
      WRITE(6,650) TIME
  650 FORMAT(//10X,'ALGORITHM TIME =',F10.4,' SECONDS')
      GO TO 10
C  IF OLD COMBINATIONS HAVE ALREADY BEEN CHECKED AGAINST
C  AN OLD CST .LE. THE RELATIVE COST OF THE TOUR JUST
C  FOUND, GO TO 80.
   58 IF(CST.LE.CSTHLD) GO TO 80
   60 IF(J.LE.2) GO TO 80
C  PROCEEDING BACKWARD THROUGH THE RANKED, NONBASIC
C  VARIABLES, RECHECK OLD COMBINATIONS AGAINST THE NEW CST
C  WHICH IS THE RELATIVE COST OF THE LATEST TOUR FOUND.
      KJ1=J-1
      DO 70 K=2,KJ1
      J=KJ1-K+2
      J1=J-1
      ICJ=IC(J)
      L1=LEG(1,ICJ)
      L2=LEG(2,ICJ)
      IB(M+1)=ICJ
      KB(M+1)=1
      WRITE(6,630) ICJ
      WRITE(6,620) J1
C  WHETHER OR NOT A COMBINATION IS FOUND TO PROVIDE
C  A TOUR, CONTINUE.
      CALL LINCOM(&70,&10)
   70 CONTINUE
C  STOP CLOCK.
   80 CALL TIMEIT(-1,TIME)
      WRITE(6,660) KZ
  660 FORMAT(//10X,'** Z =',I5,' IS OPTIMAL **')
      WRITE(6,650) TIME
      GO TO 10
   90 STOP
      END
```

```
      SUBROUTINE PRINTC
      COMMON /TSBLK/ COST,CSTLC(1600),NLC(1000),
     1               IC(1600),KC(40),KB(120),IAX(40),
     2               IROW(40),ICOL(40),LC(40),
     3               KZ,NC,J,ICJ,L1,L2,J1,CST,XZ
      COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
     1               LEG(2,1600),IB(120),MM,M,N
      WRITE(6,600)
  600 FORMAT('1',11X,'COST MATRIX')
      WRITE(6,610) (I,I=1,MM)
  610 FORMAT(//11X,40I3)
      WRITE(6,620)
  620 FORMAT(' ')
      I=0
      DO 20 K=1,N,MM
      L=K+MM-1
      J=0
      DO 10 LL=K,L
      J=J+1
      KC(J)=C(LL)
   10 CONTINUE
      I=I+1
      WRITE(6,630) I,(KC(J),J=1,MM)
  630 FORMAT( 8X,I2,1X,40I3)
   20 CONTINUE
      DO 30 J=1,N
   30 CSTLC(J)=C(J)
      RETURN
      END


      SUBROUTINE PRINTA
      COMMON /TSBLK/ COST,CSTLC(1600),NLC(1000),
     1               IC(1600),KC(40),KB(120),IAX(40),
     2               IROW(40),ICOL(40),LC(40),
     3               KZ,NC,J,ICJ,L1,L2,J1,CST,XZ
      COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
     1               LEG(2,1600),IB(120),MM,M,N
      WRITE(6,600)
  600 FORMAT('1',10X,'OPTIMAL ASSIGNMENT SOLUTION')
      IF(M.EQ.(2*MM)) GO TO 10
      WRITE(6,610)
  610 FORMAT(19X,'(MODIFIED)')
      GO TO 60
   10 KZ=Z
      DO 20 I=1,M
   20 KB(I)=H(I)+.5
      NC=0
      MM1=MM+1
      DO 50 J=1,N
      DO 30 I=1,M
      IF(J.EQ.IB(I)) GO TO 50
   30 CONTINUE
      DO 40 K=1,N,MM1
      IF(J.EQ.K) GO TO 50
   40 CONTINUE
      NC=NC+1
      IC(NC)=J
   50 CONTINUE
   60 IF(MM.LE.7) GO TO 80
      WRITE(6,620)
  620 FORMAT(//10X,'BASIS  B'//)
      DO 70 I=1,M
      WRITE(6,630) IB(I), KB(I)
  630 FORMAT(10X,I4,2X,I2)
   70 CONTINUE
      WRITE(6,640) KZ
  640 FORMAT(/15X,'Z=',I4)
```

```
      RETURN
   80 WRITE(6,650)
  650 FORMAT(//22X,'COEFF VECTORS')
      WRITE(6,660) (IC(J),J=1,NC)
  660 FORMAT(/10X,'BASIS  B  ',37I3)
      WRITE(6,670)
  670 FORMAT(' ')
      DO 100 I=1,M
      DO 90 J=1,NC
      XXHOLD=B(I,LEG(1,IC(J)))+B(I,LEG(2,IC(J)))
      IAX(J)=XXHOLD+SIGN(.5,XXHOLD)
   90 CONTINUE
      WRITE(6,680) IB(I), KB(I), (IAX(J),J=1,NC)
  680 FORMAT(10X,I4,2X,I2,2X,37I3)
  100 CONTINUE
      DO 110 J=1,NC
  110 KC(J)=C(IC(J))
      WRITE(6,690) (KC(J),J=1,NC)
  690 FORMAT(/10X,'REL COST',2X,37I3)
      WRITE(6,640) KZ
      RETURN
      END


      SUBROUTINE SOLCHK(*)
      COMMON /TSBLK/ COST,CSTLC(1600),NLC(1000),
     1               IC(1600),KC(40),KB(120),IAX(40),
     2               IROW(40),ICOL(40),LC(40),
     3               KZ,NC,J,ICJ,L1,L2,J1,CST,XZ
      COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
     1               LEG(2,1600),IB(120),MM,M,N
C  CONVERT SUBSCRIPTS OF BASIC VARIABLES EQUAL TO ONE
C  BACK TO ORIGINAL DOUBLE-SUBSCRIPTED FORM.
      I=0
      DO 20 L=1,MM
   10 I=I+1
      IF(KB(I).EQ.0) GO TO 10
      IROW(L)=LEG(1,IB(I))
      ICOL(L)=LEG(2,IB(I))-MM
   20 CONTINUE
C  COUNT THE NUMBER OF STEPS REQUIRED
C  TO GO FROM ORIGIN BACK TO ORIGIN.
      KSTEP=1
      K=1
   30 DO 40 L=2,MM
      IF(IROW(L).NE.ICOL(K)) GO TO 40
      KSTEP=KSTEP+1
      IF(ICOL(L).EQ.IROW(1)) GO TO 50
      K=L
      GO TO 30
   40 CONTINUE
   50 IF(KSTEP.LT.MM) RETURN
      CST=COST
      XZ=Z+COST
      KZ=XZ
      WRITE(6,610)
  610 FORMAT(//1CX,'A TRAVELING SALESMAN SOLUTION IS --'//)
      DO 60 L=1,MM
      WRITE(6,620) IROW(L),ICOL(L)
  620 FORMAT(21X,'X(',I2,',',I2,') = 1')
   60 CONTINUE
      WRITE(6,630) KZ
  630 FORMAT(/28X,'Z =',I5)
      RETURN 1
      END
```

```
      SUBROUTINE RANKC
      COMMON /TSBLK/ COST,CSTLC(1600),NLC(1000),
     1               IC(1600),KC(40),KB(120),IAX(40),
     2               IROW(40),ICOL(40),LC(40),
     3               KZ,NC,J,ICJ,L1,L2,J1,CST,XZ
      COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
     1               LEG(2,1600),IB(120),MM,M,N
      NC1=NC-1
      DO 20 K=1,NC1
      CMIN=C(IC(K))
      LMIN=K
      K1=K+1
      DO 10 L=K1,NC
      IF(C(IC(L)).GE.CMIN) GO TO 10
      CMIN=C(IC(L))
      LMIN=L
   10 CONTINUE
      ICHOLD=IC(K)
      IC(K)=IC(LMIN)
      IC(LMIN)=ICHOLD
   20 CONTINUE
      RETURN
      END


      SUBROUTINE REPART(*)
      COMMON /TSBLK/ COST,CSTLC(1600),NLC(1000),
     1               IC(1600),KC(40),KB(120),IAX(40),
     2               IROW(40),ICOL(40),LC(40),
     3               KZ,NC,J,ICJ,L1,L2,J1,CST,XZ
      COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
     1               LEG(2,1600),IB(120),MM,M,N
      IIHOLD=0
      DO 120 II=1,M
C   IB IS VECTOR OF BASIC-VARIABLE SUBSCRIPTS.
C   NEGATIVE VALUES INDICATE ARTIFICIALS.
      IF(IB(II).GT.0) GO TO 120
      DO 10 J=1,NC
      ICJ=IC(J)
      L1=LEG(1,ICJ)
      L2=LEG(2,ICJ)
      XHLD=B(II,L1)+B(II,L2)
      IPIV=XHLD+SIGN(.5,XHLD)
      IF(IPIV.NE.0) GO TO 30
   10 CONTINUE
C   IF ONE ARTIFICIAL CAN NOT BE REPLACED WITH A
C   PIVOT OPERATION, EARMARK ITS ROW FOR DELETION.
C   IF THERE IS NO PIVOT ELEMENT FOR MORE THAN ONE
C   ARTIFICIAL, TERMINATE EXECUTION ON THIS PROBLEM.
      IF(IIHOLD.NE.0) GO TO 20
      IIHOLD=II
      GO TO 120
   20 WRITE(6,600)
  600 FORMAT(//1CX,'CAN NOT REPLACE ARTIFICIALS')
      RETURN 1
C  REPLACE ARTIFICIAL WITH A PIVOT OPERATION.
   30 IB(II)=ICJ
      DO 50 I=1,M
      IF(I.EQ.II) GO TO 50
      XHLD=B(I,L1)+B(I,L2)
      IYIJ=XHLD+SIGN(.5,XHLD)
      IF(IYIJ.EQ.0) GO TO 50
      IQ=IYIJ/IPIV
      DO 40 L=1,M
   40 B(I,L)=B(I,L)-IQ*B(II,L)
   50 CONTINUE
      IF(IPIV.EQ.1) GO TO 70
      DO 60 L=1,M
```

```
   60 B(II,L)=(-1.)*B(II,L)
   70 NC=NC-1
      IF(J.EQ.NC+1) GO TO 90
C  SHIFT NONBASIC VECTORS LEFTWARD TO FILL GAP.
      DO 80 L=J,NC
   80 IC(L)=IC(L+1)
C  RECOMPUTE RELATIVE COSTS.
   90 DO 110 J=1,NC
      ICJ=IC(J)
      L1=LEG(1,ICJ)
      L2=LEG(2,ICJ)
      CHOLD=CSTLC(ICJ)
      DO 100 I=1,M
      IF(IB(I).LT.0) CX=1000000.
      IF(IB(I).GT.0) CX=CSTLC(IB(I))
  100 CHOLD=CHOLD-CX*(B(I,L1)+B(I,L2))
  110 C(ICJ)=CHOLD
      CALL RANKC
  120 CONTINUE
C  FIND A REDUNDANT TABLEAU ROW TO DROP.
      M1=M
      M=M-1
      IF(IIHOLD.NE.0) GO TO 150
      DO 140 II=1,M1
      DO 130 J=1,NC
      ICJ=IC(J)
      L1=LEG(1,ICJ)
      L2=LEG(2,ICJ)
      XHLD=B(II,L1)+B(II,L2)
      IYIJ=XHLD+SIGN(.5,XHLD)
      IF(IYIJ.NE.0) GO TO 140
  130 CONTINUE
      IIHOLD=II
      GO TO 150
  140 CONTINUE
      RETURN
  150 IF(IIHOLD.EQ.M1) RETURN
C  SHIFT TABLEAU ROWS UPWARD TO FILL GAP.
      DO 170 I=IIHOLD,M
      IB(I)=IB(I+1)
      H(I)=H(I+1)
      DO 160 J=1,M1
  160 B(I,J)=B(I+1,J)
  170 CONTINUE
      RETURN
      END


      SUBROUTINE LINCOM(*,*)
      COMMON /TSBLK/ COST,CSTLC(1600),NLC(1000),
     1               IC(1600),KC(40),KB(120),IAX(40),
     2               IROW(40),ICOL(40),LC(40),
     3               KZ,NC,J,ICJ,L1,L2,J1,CST,XZ
      COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
     1               LEG(2,1600),IB(120),MM,M,N
      IF(J.LE.32) GO TO 20
      WRITE(6,600)
  600 FORMAT(15X,'CAN NOT PROCEED BECAUSE LN COM COUNT ',
     1       'WILL EXCEED 2**31 - 1')
   10 CALL TIMEIT(-1,TIME)
      WRITE(6,610)
  610 FORMAT(//15X,'PROGRAM LIMITATIONS FORCE ',
     1       'ALGORITHM HALT')
      WRITE(6,620) TIME
  620 FORMAT(/15X,'ALGORITHM TIME =',F10.4,' SECONDS')
      RETURN 2
C  INITIALIZE COUNTERS.
   20 NGEN=0
      NCSTF=0
      NCNF=0
      KOST=CST
C  INITIALIZE COMBINATION GENERATER.
```

```
      DO 30 I=1,J1
   30 LC(I)=0
C  BEGIN GENERATING.
C  J1 IS EQUAL TO J-1 AND IS THE TOTAL NUMBER OF
C  NONBASIC VARIABLES TO BE TRIED IN COMBINATION
C  WITH THE J TH NONBASIC VARIABLE.
C  KT IS THE NUMBER OF BINARY NUMBERS WITH EXACTLY
C  IJ1 DIGITS WHERE THE LEFTMOST DIGIT IS ALWAYS A ONE.
      DO 170 IJ1=1,J1
      KT=2**(IJ1-1)
      DO 160 II=1,KT
      DO 40 I=1,IJ1
      IF(LC(I).EQ.0) GO TO 50
   40 LC(I)=0
   50 LC(I)=1
C  INCREMENT THE NUMBER OF COMBINATIONS GENERATED BY ONE.
      NGEN=NGEN+1
C  COMPUTE LATEST COMBINATION'S RELATIVE COST.
      COST=C(ICJ)
      DO 60 I=1,IJ1
      IF(LC(I).EQ.1) COST=COST+C(IC(I))
   60 CONTINUE
      IF(COST.LE.CST) GO TO 110
C  THE FOLLOWING SEQUENCE OF STATEMENTS DOWN TO
C  STATEMENT 110 ARE VARIOUS CHECKS TO SEE IF
C  ENTIRE BLOCKS OF COMBINATIONS CAN BE SKIPPED.
      IF(II.GT.1) GO TO 70
      IF(NCNF.GT.0) GO TO 180
   65 WRITE(6,640) NGEN,KOST,NCSTF,NCNF
  640 FORMAT(15X,'NR LN COM NECESSARY TO GENERATE =',I10,
     1       /15X,'NR WITH REL COST .LE.',I4,' =',6X,I10,
     2       /15X,'NR FOUND CONSTRAINT FEASIBLE =',3X,I10)
      RETURN
   70 DO 80 I=2,IJ1
      IF(II.EQ.2**(I-2)+1) GO TO 90
   80 CONTINUE
      GO TO 160
   90 DO 100 I=1,IJ1
  100 LC(I)=1
      GO TO 170
C  INCREMENT BY ONE THE NUMBER OF COMBINATIONS
C  FOUND LESS THAN OR EQUAL TO THE UPPER BOUND.
  110 NCSTF=NCSTF+1
C  CHECK THE LATEST COMBINATION FOR CONSTRAINT FEASIBILITY.
      DO 130 K=1,M
      XHOLD=H(K)-B(K,L1)-B(K,L2)
      KB(K)=XHOLD+SIGN(.5,XHOLD)
      DO 120 I=1,IJ1
      IF(LC(I).EQ.0) GO TO 120
      ICI=IC(I)
      XHOLD=KB(K)-B(K,LEG(1,ICI))-B(K,LEG(2,ICI))
      KB(K)=XHOLD+SIGN(.5,XHOLD)
  120 CONTINUE
      IF(KB(K).NE.0.AND.KB(K).NE.1) GO TO 160
  130 CONTINUE
C  INCREMENT BY ONE THE NUMBER OF COMBINATIONS
C  FOUND CONSTRAINT FEASIBLE.
      NCNF=NCNF+1
      IF (NCNF.LE.1000) GO TO 140
      WRITE(6,640) NGEN,KOST,NCSTF,NCNF
      WRITE(6,650)
  650 FORMAT(15X,'CAN NOT PROCEED BECAUSE STORAGE IS ',
     1       'ALLOCATED FOR NO MORE THAN 1000 ',
     2       'CN FEAS LN COM')
      GO TO 10
C  STORE THE RELATIVE COST AND IDENTIFYING NUMBER
C  FOR THE LATEST CONSTRAINT FEASIBLE COMBINATION.
  140 CSTLC(NCNF)=COST
      NLC(NCNF)=2**(IJ1-1)+II-1
      IF(NCNF.EQ.1) GO TO 160
C  ARRANGE VALUES IN CSTLC AND NLC ACCORDING TO
C  ASCENDING ORDER OF RELATIVE COSTS.
```

```
      NCNF1=NCNF-1
      DO 150 IL=1,NCNF1
      LI=NCNF1-IL+1
      IF(CSTLC(LI+1).GE.CSTLC(LI)) GO TO 160
      CHOLD=CSTLC(LI)
      NHOLD=NLC(LI)
      CSTLC(LI)=CSTLC(LI+1)
      NLC(LI)=NLC(LI+1)
      CSTLC(LI+1)=CHOLD
  150 NLC(LI+1)=NHOLD
  160 CONTINUE
  170 CONTINUE
      IF(NCNF.EQ.0) GO TO 65
C  CHECK EACH COMBINATION STORED AND RANKED ABOVE
C  FOR A TRAVELING-SALESMAN SOLUTION.
  180 WRITE(6,640) NGEN,KOST,NCSTF,NCNF
      DO 210 I=1,NCNF
C  RECOMPUTE OPTIMAL ASSIGNMENT BASIC-VARIABLE VALUES.
      DO 185 K=1,M
      XHOLD=H(K)-B(K,L1)-B(K,L2)
  185 KB(K)=XHOLD+SIGN(.5,XHOLD)
      L=0
      KM=M+1
      NL=NLC(I)
C  REGENERATE A COMBINATION STORED ABOVE.
  190 L=L+1
      LCL=MOD(NL,2)
      IF(LCL.EQ.0) GO TO 200
      ICL=IC(L)
C  RECOMPUTE NEW BASIC-VARIABLE VALUES
C  FOR LATEST REGENERATED COMBINATION.
      DO 195 K=1,M
      XHOLD=KB(K)-B(K,LEG(1,ICL))-B(K,LEG(2,ICL))
  195 KB(K)=XHOLD+SIGN(.5,XHOLD)
C  AUGMENT ASSIGNMENT BASIS WITH THE NONBASIC
C  VARIABLES IN THE LATEST COMBINATION.
      KM=KM+1
      IB(KM)=ICL
      KB(KM)=1
  200 NL=NL/2
      IF(NL.GT.0) GO TO 190
C  CHECK LATEST COMBINATION FOR A TRAVELING-SALESMAN TOUR.
      COST=CSTLC(I)
      CALL SOLCHK(&220)
C  AS SOON AS A TOUR IS FOUND GO TO STATEMENT 220,
C  OTHERWISE CONTINUE.
  210 CONTINUE
      RETURN
  220 RETURN 1
      END


      SUBROUTINE ASSIGN(*)
      COMMON /ASGBLK/ D(1600),DB( 80),DUAL( 80),AX( 80),
     1                E,AD,W,INDEX,K1,JS,IR,ID,ITR
      COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
     1               LEG(2,1600),IB(120),MM,M,N
      WRITE(6,35)
   35 FORMAT('1')
      K1=2
      INDEX=0
      Z=0.
      NN=MM
      DO 4 I=1,M
    4 H(I)=1.
      DO 1 I=1,MM
      U=10000.
      V=10000.
      KA=NN*I-NN+1
      KB=NN*I
      DO 6 J=KA,KB
      IF(C(J).LT.U) U=C(J)
```

```
6       LEG(1,J)=I
        IF(U.EQ.0.) GO TO 14
        DO 27 J=KA,KB
27      C(J)=C(J)-U
        Z=Z+U*H(I)
14      K=I+MM
        DO 3 J=I,N,NN
        IF(C(J).LT.V) V=C(J)
3       LEG(2,J)=K
        IF(V.EQ.0.) GO TO 1
        DO 28 J=I,N,NN
28      C(J)=C(J)-V
        Z=Z+V*H(K)
1       CONTINUE
        CALL LIP(&33)
        RETURN
   33   RETURN 1
        END


        SUBROUTINE LIP(*)
        COMMON /ASGBLK/ D(1600),DB( 80),DUAL( 80),AX( 80),
       1                E,AD,W,INDEX,K1,JS,IR,ID,ITR
        COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
       1               LEG(2,1600),IB(120),MM,M,N
        E=.5
        DO 5 J=1,N
5       D(J)=-2.
        DO 3 J=1,M
        DO 2 I=1,M
2       B(I,J)=0.
3       B(J,J)=1.
        DO 4 I=1,M
        DB(I)=-1.
        DUAL(I)=0.
4       IB(I)=-I
        AD=1.
        ID=1
        W=0.
        DO 6 I=1,M
6       W=W+H(I)
        ITR=0
7       IF(ABS(W)-E)100,100,8
8       DO 9 J=1,N
        IF(D(J))10,9,9
9       CONTINUE
        GO TO 101
10      CX=100000.
        DO 11 J=1,N
        JT=J
        IF(D(J))12,11,11
12      IF(C(J)-.0001.LT.0.) GO TO 41
        IF(CX+C(J)/D(J))11,11,13
13      CX=-C(J)/D(J)
        JS=J
11      CONTINUE
        DO 14 J=1,N
14      C(J)=C(J)+CX*D(J)
        C(JS)=0.
        Z=Z+CX*W
        JT=JS
        DO 40 I=1,M
40      DUAL(I)=DUAL(I)-CX*DB(I)
15      IF(W-E)100,100,41
41      DX=0.
        DO 16 J=JT,N
        IF(C(J)-0.0001)17,17,16
17      C(J)=0.
        IF(D(J).GE.DX) GO TO 16
        DX=D(J)
19      JS=J
        IF(D(JS).EQ.-2.) GO TO 1
```

```
16     CONTINUE
1      WRITE(6,213) JS,D(JS)
213    FORMAT(1H ,3HJS=,I4,3X,2HD=,F8.4)
       IF(D(JS))20,7,7
20     CALL ELEM
       IF(ID-1)528,26,26
26     CALL PIVOT(&33)
       GO TO 15
100    CONTINUE
       INFEA =0
       GO TO 425
101    WRITE (6,205)
205    FORMAT(1H ,10HINFEASIBLE)
       INFEA =1
33     RETURN 1
425    CONTINUE
528    I=1
       RETURN
       END


       SUBROUTINE ELEM
       COMMON /ASGBLK/ D(1600),DB( 80),DUAL( 80),AX( 80),
      1                 E,AD,W,INDEX,K1,JS,IR,ID,ITR
       COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
      1               LEG(2,1600),IB(120),MM,M,N
       IR=0
       J=JS
       DO 21 I=1,M
       AX(I)=0.
       DO 21 K=1,2
       K2=LEG(K,J)
       AX(I)=AX(I)+B(I,K2)
    21 CONTINUE
       EE=E
       DO 25 I=1,M
       IF(AX(I)-E) 25,25,810
   810 IF(H(I)-E) 82,82,23
    82 IR=I
       GO TO 13
    23 IF(IR.EQ.0) GO TO 24
       IF(ABS(H(I)-AX(I)*X)-EE) 70,70,2
     2 IF(H(I)-AX(I)*X) 24,70,25
    24 X=H(I)/AX(I)
     3 IR=I
       EE=E/AX(IR)
       GO TO 25
    70 IF(IB(I)) 74,25,73
    73 IF(IB(IR)) 25,25,4
    74 IF(IB(IR)) 4,25,3
     4 IF(AX(IR)-AX(I)) 25,25,3
    25 CONTINUE
    13 RETURN
       END


       SUBROUTINE PIVOT(*)
       COMMON /ASGBLK/ D(1600),DB( 80),DUAL( 80),AX( 80),
      1                 E,AD,W,INDEX,K1,JS,IR,ID,ITR
       COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
      1               LEG(2,1600),IB(120),MM,M,N
84     WRITE (6,214) IR,AX(IR)
214    FORMAT(1H+,25X,3HIR=,I3,3X,3HAX=,F8.4)
       IB(IR)=JS
       AD=AD*ABS(AX(IR))
       DI=ID
       ID=INT(DI*ABS(AX(IR))+0.5)
       DI=ID
       IF(AX(IR).LT.1.+E.AND.AX(IR).GT.1.-E) GO TO 1
       HX=1./AX(IR)
       H(IR)=H(IR)*HX
       DO 11 J=1,M
```

```
11      B(IR,J)=B(IR,J)*HX
1       DO 3 I=1,M
        IF(I.EQ.IR) GO TO 3
        IF(ABS(AX(I)).LT.E) GO TO 3
        IF(AX(I).LT.1.+E.AND.AX(I).GT.1.-E) GO TO 4
        IF(AX(I).GT.-1.-E.AND.AX(I).LT.-1.+E) GO TO 5
        DO 29 J=1,M
29      B(I,J)=B(I,J)-AX(I)*B(IR,J)
        H(I)=H(I)-AX(I)*H(IR)
        GO TO 3
4       DO 6 J=1,M
6       B(I,J)=B(I,J)-B(IR,J)
        H(I)=H(I)-H(IR)
        GO TO 3
5       DO 7 J=1,M
7       B(I,J)=B(I,J)+B(IR,J)
        H(I)=H(I)+H(IR)
3       CONTINUE
        W=W+D(JS)*H(IR)
        DO 8 J=1,M
8       DB(J)=DB(J)-D(JS)*B(IR,J)
        E=0.5/DI
        DO 90 K=25,600,25
        IF(K-ITR)90,91,95
90      CONTINUE
        GO TO 95
91      DO 92 J=1,M
        DB(J)=SIGN(AINT(DI*ABS(DB(J))+0.5)/DI,DB(J))
        DO 92 I=1,M
92      B(I,J)=SIGN(AINT(DI*ABS(B(I,J))+0.5)/DI,B(I,J))
        DO 93 I=1,M
93      H(I)=SIGN(AINT(DI*ABS(H(I))+0.5)/DI,H(I))
        W=AINT(DI*W+0.5)/DI
95      DO 34 J=1,N
        D(J)=0.
        DO 33 K=1,K1
        K2=LEG(K,J)
        IF(K2.EQ.0) GO TO 33
        D(J)=D(J)+DB(K2)
33      CONTINUE
        IF(ABS(D(J))-E)35,35,34
35      D(J)=0.
34      CONTINUE
        ITR=ITR+1
        WRITE (6,210) ITR,W,Z,ID,AD
 210    FORMAT(1H ,5X,4HITR=,I5,5X,2HW=,F12.8,5X,2HZ=,
     1    F12.5,5X,3HID=,I6,3X,3HAD=,F12.4)
        IF(ITR.EQ.300)GO TO 21
        RETURN
21      WRITE(6,220)
220     FORMAT(//6X,'ITERATION LIMIT EXCEEDED')
        RETURN 1
        END


        SUBROUTINE RAND(*)
        COMMON /SHARE/ C(1600),B( 80, 80),H( 80),Z,
     1                 LEG(2,1600),IB(120),MM,M,N
        DATA I/3125/
        DO 10 J=1,N
        I=MOD(3125*I,2048)
        X=(100.*I)/2048. + .5
        K=X
        C(J)=K
   10   CONTINUE
        MM1=MM+1
        DO 20 J=1,N,MM1
        C(J)=999.
   20   CONTINUE
        RETURN 1
        END
```

```
      SUBROUTINE TIMEIT(N,TIME)
C
C     N=0 STARTS CLOCK; N=-1 STOPS CLOCK
C
      IT=N+2
      GO TO (20,10),IT
   10 CALL TIMON(M)
      TIMEM=M
      RETURN
   20 CALL TIMOFF(M)
      TIME=M
      TIME=(TIMEM-TIME)*.000026
      RETURN
      END
C  PLACE A /* CARD HERE.
//ASM.SYSIN DD *
TIMEALL  CSECT
         ENTRY  TIMON,TIMOFF
TIMON    SAVE   (14,12)          ENTRY VIA -CALL TIMON(N)- 15.5
         USING  TIMON,12
         LR     12,15
         ST     13,TEMP1
         LA     13,SAVE1
         L      2,0(1,0)
         L      3,TOTIME
         ST     3,CLOCKR
         ST     3,0(2,0)
         STIMER TASK,TUINTVL=CLOCKR
         L      13,TEMP1
EXIT     RETURN (14,12),T,RC=0
TIMOFF   SAVE   (14,12)          ENTER VIA -CALL TIMOFF(N)-
         USING  TIMOFF,12
         LR     12,15
         ST     13,TEMP1
         LA     13,SAVE1
         L      2,0(1,0)
         TTIMER CANCEL
         ST     0,0(2,0)
         L      13,TEMP1
         RETURN (14,12),T,RC=0
         CNOP   0,4
TOTIME   DC     X'7FFFFFFF'
CLOCKR   DS     F
SAVE1    DS     18F
TEMP1    DS     F
         END
C  PLACE A /* CARD HERE.
//GO.FT06F001  DD  SPACE=(CYL,(10,1))
//GO.SYSIN DD *
C  PLACE DATA CARDS HERE.
C  PLACE AN ORANGE /* CARD HERE.
```

APPENDIX B COMPUTER OUTPUT FOR A SELECTED PROBLEM

COST MATRIX[2]

	1	2	3	4	5	6	7	8	9	10
1	999	24	18	22	31	19	33	25	30	26
2	15	999	19	27	26	32	25	31	28	18
3	22	23	999	23	16	29	27	18	16	27
4	24	31	18	999	19	13	28	9	19	27
5	23	18	34	20	999	31	24	15	25	8
6	24	12	17	15	10	999	11	16	21	31
7	28	15	27	35	19	18	999	21	21	19
8	13	24	18	13	13	22	25	999	29	24
9	17	21	18	24	27	24	34	31	999	18
10	18	19	29	16	23	17	18	31	23	999

OPTIMAL ASSIGNMENT SOLUTION

BASIS B

82	0
11	1
75	1
38	1
50	1
29	1
3	0
94	1
6	1
81	0
74	0
62	1
83	1
4	0
55	0
69	0
57	1
48	0
90	0

$z = 140$

---

[2] Alternate solution procedures for this ten-city problem are discussed in References 8 and 9.

```
TRY X( 8) = 1  REL COST = 0

TRY X(97) = 1  REL COST = 1
TRY THE (2** 1) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  1
     NR WITH REL COST .LE.  2  =         1
     NR FOUND CONSTRAINT FEASIBLE  =     0

TRY X(68) = 1 REL COST = 2
TRY THE (2** 2) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  2  =         1
     NR FOUND CONSTRAINT FEASIBLE  =     0

TRY X(99) = 1  REL COST = 2
TRY THE (2** 3) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE. 2  =          1
     NR FOUND CONSTRAINT FEASIBLE  =     0

TRY X(20) = 1  REL COST = 2
TRY THE (2** 4) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  2  =         1
     NR FOUND CONSTRAINT FEASIBLE  =     0

TRY X(84) = 1  REL COST = 2
TRY THE (2** 5) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  3  =         3
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X(20) AND
TRY THE (2** 4) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  3  =         3
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X(99) AND
TRY THE (2** 3) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  3  =         3
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X(68) AND
TRY THE (2** 2) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  3
     NR WITH REL COST .LE.  3  =         3
     NR FOUND CONSTRAINT FEASIBLE  =     0
```

```
RETURN TO X(97) AND
TRY THE (2** 1) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  1
     NR WITH REL COST .LE.  3  =        1
     NR FOUND CONSTRAINT REASIBLE  =    0

TRY X(58) = 1 REL COST = 3
TRY THE (2** 6) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  3  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X( 2) = 1  REL COST = 3
TRY THE (2** 7) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  3  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X( 9) = 1  REL COST = 3
TRY THE (2** 8) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  3  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(19) = 1  REL COST = 3
TRY THE (2** 9) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  3 =         1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(13) = 1  REL COST = 3
TRY THE (2**10) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  3  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(52) = 1  REL COST = 3
TRY THE (2**11) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  3  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(65) = 1  REL COST = 3
TRY THE (2**12) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  4  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(52) AND
TRY THE (2**11) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  4  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0
```

```
RETURN TO X(13) AND
TRY THE (2**10) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  4  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    1

RETURN TO X(19) AND
TRY THE (2** 9) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  4  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X( 9) AND
TRY THE (2** 8) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  4  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X( 2) AND
TRY THE (2** 7) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  4  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(58) AND
TRY THE (2** 6) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  4  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(84) AND
TRY THE (2** 5) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  = 12
     NR WITH REL COST .LE.  4  =        9
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(20) AND
TRY THE (2** 4)  - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  9
     NR WITH REL COST .LE.  4  =        7
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(99) AND
TRY THE (2** 3) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  6
     NR WITH REL COST .LE.  4  =        5
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(68) AND
TRY THE (2** 2) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  3
     NR WITH REL COST .LE.  4  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0
```

```
RETURN TO X(97) AND
TRY THE (2** 1) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  1
     NR WITH REL COST .LE.  4  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(28) = 1  REL COST = 4
TRY THE (2**13) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  4  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(96) = 1  REL COST = 4
TRY THE (2**14) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  4  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(17) = 1  REL COST = 4
TRY THE (2**15) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  4  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(92) = 1  REL COST = 4
TRY THE (2**16) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  5  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(17) AND
TRY THE (2**15) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  5  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(96) AND
TRY THE (2**14) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  5  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(28) AND
TRY THE (2**13) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  5  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(65) AND
TRY THE (2**12) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  = 16
     NR WITH REL COST .LE.  5 =        11
     NR FOUND CONSTRAINT FEASIBLE  =    0
```

```
RETURN TO X(52) AND
TRY THE (2**11) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  16
     NR WITH REL COST .LE.  5 =          11
     NR FOUND CONSTRAINT FEASIBLE  =      0

RETURN TO X(13) AND
TRY THE (2**10) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  16
     NR WITH REL COST .LE.  5  =         11
     NR FOUND CONSTRAINT FEASIBLE  =      1

RETURN TO X(19) AND
TRY THE (2** 9) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  16
     NR WITH REL COST .LE.  5  =         11
     NR FOUND CONSTRAINT FEASIBLE  =      0

RETURN TO X( 9) AND
TRY THE (2** 8) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  = 16
     NR WITH REL COST .LE.  5  =         11
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X( 2) AND
TRY THE (2** 7) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  16
     NR WITH REL COST .LE.  5  =         11
     NR FOUND CONSTRAINT FEASIBLE  =      0

RETURN TO X(58) AND
TRY THE (2** 6) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  15
     NR WITH REL COST .LE.  5  =         11
     NR FOUND CONSTRAINT FEASIBLE  =      0

RETURN TO X(84) AND
TRY THE (2** 5) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  17
     NR WITH REL COST .LE.  5  =         15
     NR FOUND CONSTRAINT FEASIBLE  =      0

RETURN TO X(20) AND
TRY THE (2** 4) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  12
     NR WITH REL COST .LE.  5  =         11
     NR FOUND CONSTRAINT FEASIBLE  =      0

RETURN TO X(99) AND
TRY THE (2** 3) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =   7
     NR WITH REL COST .LE.  5  =          7
     NR FOUND CONSTRAINT FEASIBLE  =      0
```

```
RETURN TO X(68) AND
TRY THE (2** 2) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  3
     NR WITH REL COST .LE. 5   =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(97) AND
TRY THE (2** 1) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  1
     NR WITH REL COST .LE.  5  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(86)  =  1  REL COST = 5
TRY THE (2**17) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  5 =         1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(54) = 1  REL COST = 5
TRY THE (2**18) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  5  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(66) = 1  REL COST = 5
TRY THE (2**19) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  5  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(71) = 1  REL COST = 5
TRY THE (2**20) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  5  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

TRY X(25) = 1  REL COST = 5
TRY THE (2**21)  - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  2
     NR WITH REL COST .LE.  5  =        1
     NR FOUND CONSTRAINT FEASIBLE  =    0

     TRY X(85) = 1  REL COST = 5
TRY THE (2**22) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  6  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0

RETURN TO X(25) AND
TRY THE (2**21) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  6  =        3
     NR FOUND CONSTRAINT FEASIBLE  =    0
```

```
RETURN TO X(71) AND
TRY THE (2**20) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  6  =         3
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X(66) AND
TRY THE (2**19) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  6  =         3
     NR FOUND CONSTRAINT FEASIBLE  =     1

RETURN TO X(54) AND
TRY THE (2**18) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  6  =         3
     NR FOUND CONSTRAINT FEASIBLE  =     1

RETURN TO X(86) AND
TRY THE (2**17) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  4
     NR WITH REL COST .LE.  6  =         3
     NR FOUND CONSTRAINT FEASIBLE  =     1

RETURN TO X(92) AND
TRY THE (2**16) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  = 16
     NR WITH REL COST .LE.  6  =        11
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X(17) AND
TRY THE (2**15) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  = 16
     NR WITH REL COST .LE.  6  =        11
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X(96) AND
TRY THE (2**14) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  = 16
     NR WITH REL COST .LE.  6  =        11
     NR FOUND CONSTRAINT FEASIBLE  =     1

A TRAVELING SALESMAN SOLUTION IS --

        X( 2, 1) = 1
        X( 8, 5) = 1
        X( 4, 8) = 1
        X( 5,10) = 1
        X( 3, 9) = 1
        X( 1, 3) = 1
        X( 7, 2) = 1
        X( 6, 7) = 1
        X(10, 6) = 1
        X( 9, 4) = 1

              Z = 146
```

```
RETURN TO X(28) AND
TRY THE (2**13) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  16
     NR WITH REL COST .LE.  6  =        11
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X(65) AND
TRY THE (2**12) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  40
     NR WITH REL COST .LE.  6  =        31
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X(52) AND
TRY THE (2**11) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  37
     NR WITH REL COST .LE.  6  =        29
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X(13) AND
TRY THE (2**10) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  34
     NR WITH REL COST .LE.  6  =        27
     NR FOUND CONSTRAINT FEASIBLE  =     1

RETURN TO X(19) AND
TRY THE (2** 9) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  31
     NR WITH REL COST .LE.  6  =        25
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X( 9) AND
TRY THE (2** 8) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  28
     NR WITH REL COST .LE.  6  =        23
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X( 2) AND
TRY THE (2** 7) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  25
     NR WITH REL COST .LE.  6  =        21
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X(58) AND
TRY THE (2** 6) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  22
     NR WITH REL COST .LE.  6  =        19
     NR FOUND CONSTRAINT FEASIBLE  =     0

RETURN TO X(84) AND
TRY THE (2** 5) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  31
     NR WITH REL COST .LE.  6  =        21
     NR FOUND CONSTRAINT FEASIBLE  =     0
```

```
RETURN TO X(20) AND
TRY THE (2** 4) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =  15
     NR WITH REL COST .LE.  6  =         13
     NR FOUND CONSTRAINT FEASIBLE  =      0

RETURN TO X(99) AND
TRY THE (2** 3) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =   7
     NR WITH REL COST .LE.  6  =          7
     NR FOUND CONSTRAINT FEASIBLE  =      0

RETURN TO X(68) AND
TRY THE (2** 2) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =   3
     NR WITH REL COST .LE.  6  =          3
     NR FOUND CONSTRAINT FEASIBLE  =      0

RETURN TO X(97) AND
TRY THE (2** 1) - 1 LN COM
     NR LN COM NECESSARY TO GENERATE  =   1
     NR WITH REL COST. LE.  6  =          1
     NR FOUND CONSTRAINT FEASIBLE  =      0

**Z =  146 IS OPTIMAL  **

ALGORITHM TIME =    1.5043 SECONDS
```

LIST OF REFERENCES

1. Dantzig, George B., Linear Programming and Extensions, p. 545-547, 2d ed., Princeton University Press, 1966.

2. _____, p. 305.

3. _____, Chapters 14 and 15.

4. Ford, L. R., Jr., and Fulkerson, D. R., Flows in Networks, 3d ed., Chapter III, Princeton University Press, 1967.

5. Hadley, G., Linear Programming, 2d ed., Chapter 9, Addison-Wesley Publishing Company, Inc., 1962.

6. Hadley, G., Nonlinear and Dynamic Programming, p. 267-269, Addison-Wesley Publishing Company, Inc., 1964.

7. IBM Systems Reference Library, IBM System/360, Fortran IV Language, File No. S360-25, Form C28-6515-5, 1965, 1966.

8. Lawler, E. L. and Wood, D. E., "Branch-and-Bound Methods: A Survey," Operations Research, v. 14, p. 707-709, July-August, 1966.

9. _____, p. 709-711.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Civil Schools Branch Department of the Army Washington, D. C. 20315	1
4.	Professor Harold Greenberg, Code 55Gd Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	1
5.	Captain William W. White, USA 2040 Hanover Circle Beaumont, Texas 77706	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Postgraduate School Monterey, California 93940	Unclassified
	2b. GROUP

3. REPORT TITLE

A Traveling Salesman Algorithm

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Master's Thesis; October 1969

5. AUTHOR(S) (First name, middle initial, last name)

William Willerson White

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
October 1969	62	9

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

A possible approach to solving the traveling-salesman problem was suggested by Professor Harold Greenberg. This study developed an algorithm based on his suggestion and then programmed the algorithm for the IBM System/360. While the program solves problems of forty cities or less, it has a significant limitation. Execution is terminated on a problem if a solution is not found early enough in the trial-and-error process of the algorithm. The solution procedure developed formulates the salesman's problem as an assignment problem, obtains an optimal assignment solution, and then manipulates vectors in the final simplex tableau until an assignment solution is obtained that also satisfies the additional traveling-salesman constraints. Background to the problem is given, the algorithm is developed and stated, the computer program is described and critiqued, highlights of computational experience with the program are presented, and, finally, some conclusions and recommendations are made.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Algorithm						
Linear Programming						